Optimal Background Treatment in Pulse Dipolar Spectroscopy

Luis Fábregas Ibáñez[1], Gunnar Jeschke[1*]

[1] *ETH Zurich, Laboratory of Physical Chemistry, Vladimir-Prelog-Weg 2, 8093 Zurich, Switzerland*

Supplementary Information

1. Metric sensitivity

As mentioned in the main text, the regularized distance distributions are compared to the underlying ground truth $\mathbf{P}_0$ using three different similarity metrics: the Euclidean distance (ℓ_2)

$$D(\mathbf{P}, \mathbf{P}_0) = \|\mathbf{P} - \mathbf{P}_0\|_2, \tag{1}$$

the Bray-Curtis distance (ℓ_1)

$$D(\mathbf{P}, \mathbf{P}_0) = \frac{\|\mathbf{P} - \mathbf{P}_0\|_1}{\|\mathbf{P} + \mathbf{P}_0\|_1}, \tag{2}$$

and the Chebyshev distance (ℓ_∞)

$$D(\mathbf{P}, \mathbf{P}_0) = \|\mathbf{P} - \mathbf{P}_0\|_\infty \tag{3}$$

where $\mathbf{P}$ is the fitted distance distribution and $\mathbf{P}_0$ the ground truth. Among them, the Euclidean distance is closely related to the root-mean square deviation (RMSD), from which it differs only by a factor $\sqrt{N}\Delta r$, where N is the number of data points and Δr is the increment of the discrete distance axis. For comparison of distributions with the same N and Δr, as we perform them here, the Euclidean distance distance is proportional to the RMSD.

*Corresponding author

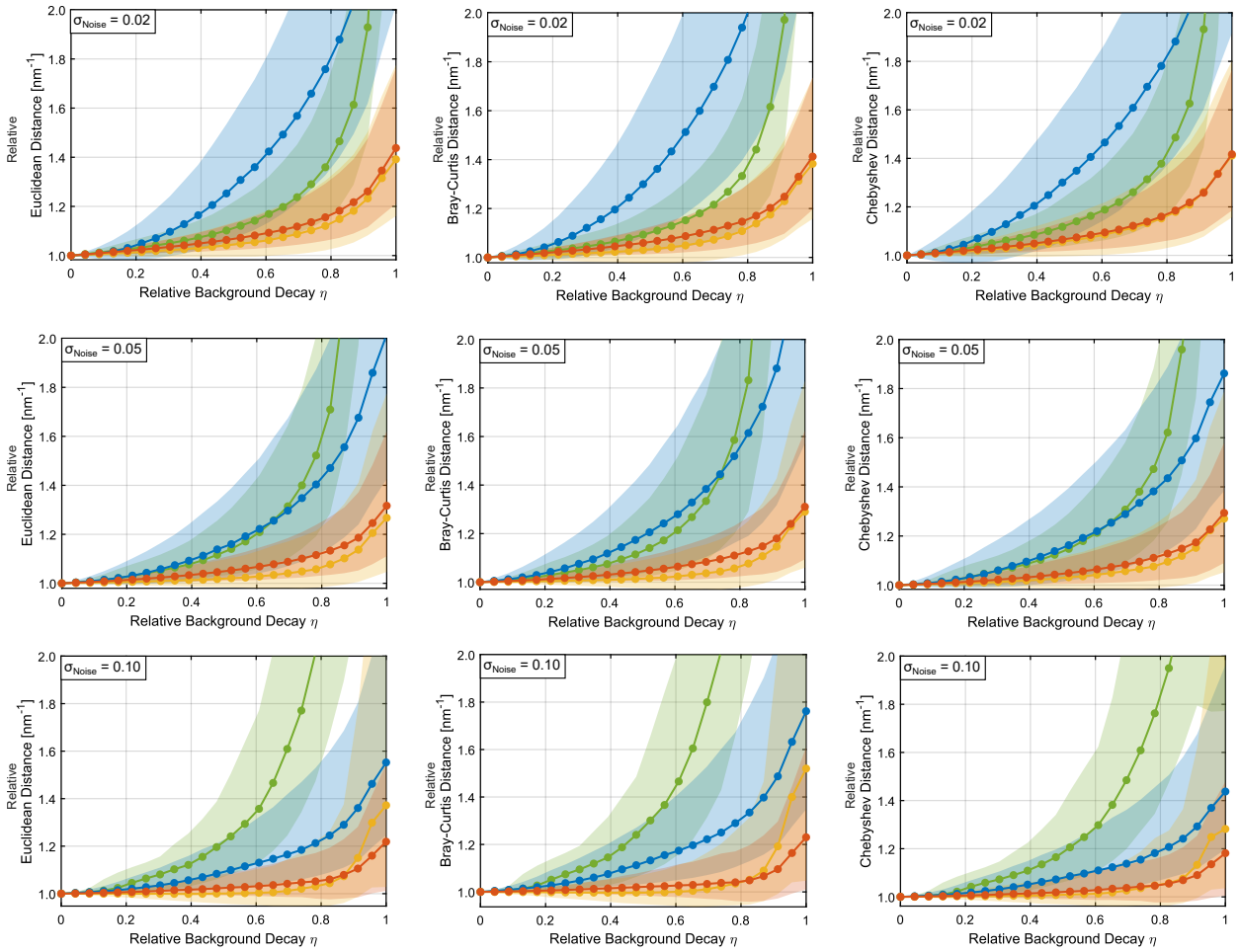

Figure S1 – Euclidean, Bray-Curtis and Chebyshev distances obtained from the statistical analysis of 8500 different form factors for different relative background decays and three different noise standard deviations: low ($\sigma = 0.02$), medium ($\sigma = 0.05$) and high ($\sigma = 0.1$) noise. For each instance the background is treated either by division (green), subtraction (blue), employing the kernel $\mathbf{K_B}$ (yellow) or kernel $\mathbf{K}_{\sqrt{B}}$ (red). All metric distances are evaluated relative to the background-free case. The coloured dotted lines represent the median values and the shaded areas, the interquartile ranges (IQR) of the data.

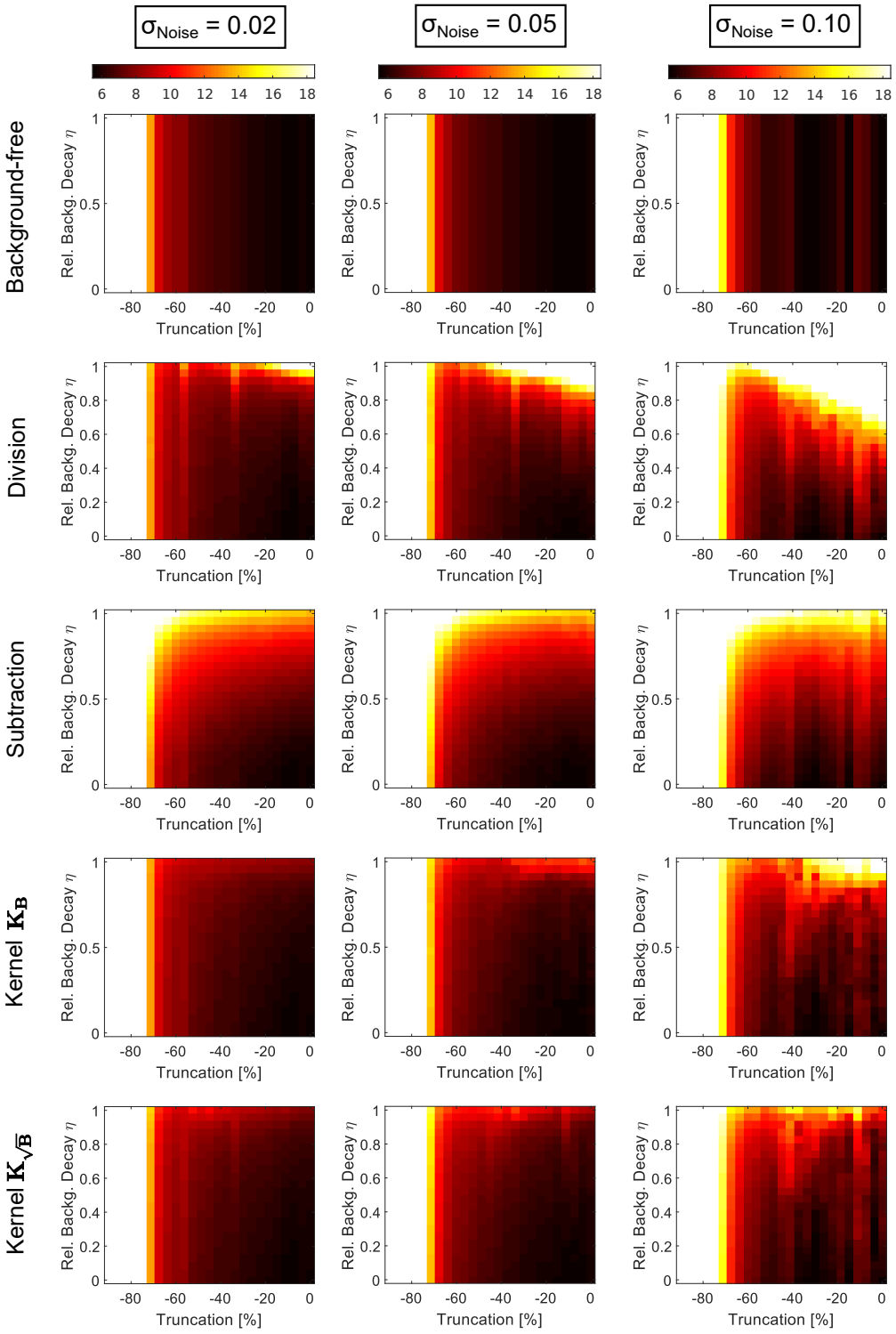

Figure S2 – Mean Euclidean distances obtained from the statistical analysis of 225 different form factors for different relative background decays and three different noise standard deviations: low ($\sigma = 0.02$), medium ($\sigma = 0.05$) and high ($\sigma = 0.1$) noise. The traces were cut at different positions relative to the end of the trace prior to regularization. Each column represents a different noise level and each row a different background treatment approach. The colour coding for each column is given by a colorbar on the top.

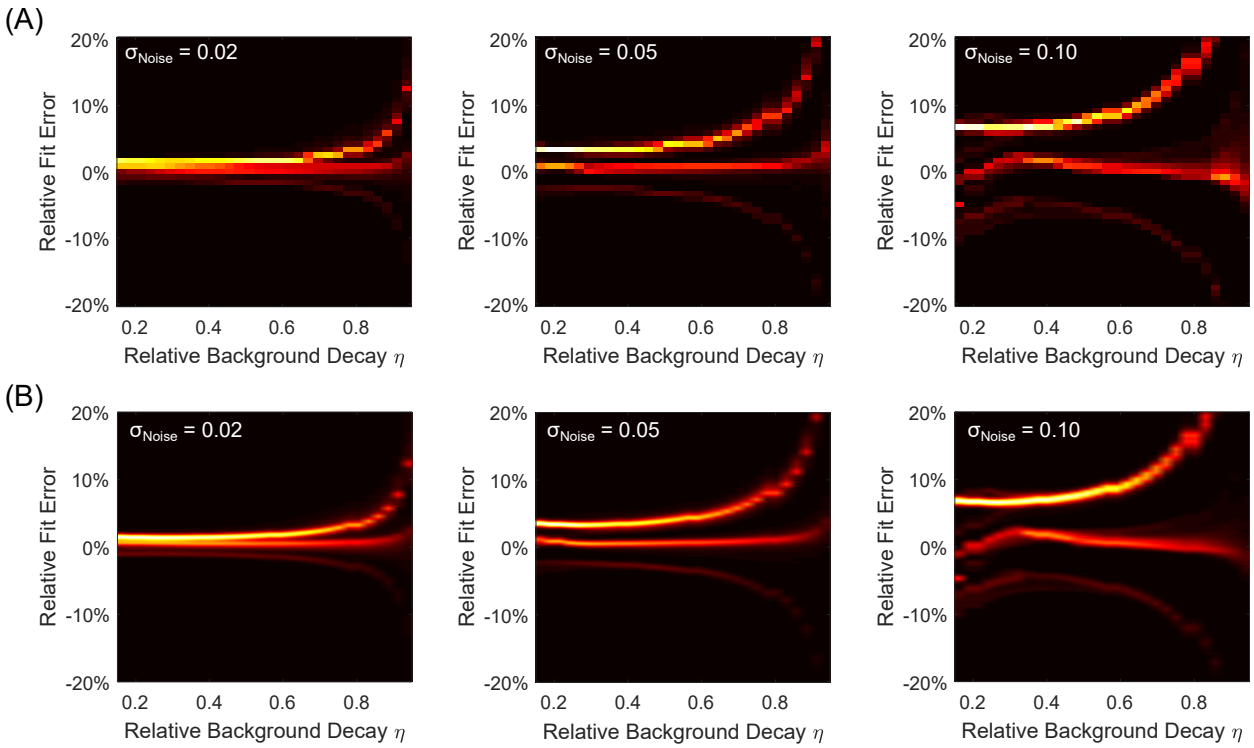

Figure S3 – Probability density distributions of fitting errors obtained from the statistical analysis of 20000 different form factors using DeerAnalysis (with default settings) for different relative background decays and three different noise standard deviations: low ($\sigma = 0.02$), medium ($\sigma = 0.05$) and high ($\sigma = 0.1$) noise. The probability density is encoded in the colour, with darker/brighter colours representing smaller/larger values, respectively. In (A) the raw probability density distributions, while in (B) the corresponding kernel density estimation of bandwidth $h = 0.30$ is given.

Table S1 – Indices of the model distance distributions employed for the examples in Fig. 9 of the main text. Given n index (*idx*), the distribution can be retrieved via $P = distributions_2LZM(idx,:)$.

Example	Distribution Index	N_{Points}	t_{max} $[\mu s]$
A	463	300	3
B	47	160	8
C	3708	160	16
D	4313	300	3
E	1203	300	16
F	4295	160	8
G	961	300	8
H	57	200	3
I	1195	300	3
J	9	100	3
K	3321	300	16

2. Model distance distribution indices

3. Methodology & Implementation

3.1. Preparing the form factors

First, we start by loading the Edwards&Stoll DEER dataset library

```
distributions_2LZM = load('distributions_2LZM');
Sdata_2LZM   = load('Sdata_2LZM');
timetraces_2LZM = load('timetraces_2LZM');
```

and, as mentioned in the main text, a subset is randomly selected from the library. Since we need to add the background prior to the noise, we take the noiseless form factors from the `timetraces_2LZM` library. The random sampling is performed using a fixed random number as follows:

```
%Generate random numbers
%-------------------------------------------------------------------------
rng(2,'twister')
RandomIndexVector = randperm(60000,Models2Test);

%Extract current random number
%-------------------------------------------------------------------------
RandomIndex= RandomIndexVector(j);

%Extract data from library
%-------------------------------------------------------------------------
ModelIndex = timetraces_2LZM.data(RandomIndex).Pidx;
%Extract time axis components
TimeStep =  timetraces_2LZM.data(RandomIndex).dt;
tmin = timetraces_2LZM.data(RandomIndex).tmin;
tmax = timetraces_2LZM.data(RandomIndex).tmax;
```

Since we need to expand the time axis to better represent RIDME and SIFTER data, we do so by increasing all `tmax` values by a fixed number (in our case `expansionFactor=2.5`).

```
%Expand time trace to longer times
%-------------------------------------------------------------------------
```

```
tmax = expansionFactor*tmax;
nPoints = tmax/TimeStep;
%Expand time trace to longer times
%-------------------------------------------------------------------------
tmax = expansionFactor*tmax;
nPoints = tmax/TimeStep;
TimeAxis = linspace(0,nPoints*TimeStep,nPoints);
```

The next step is to construct the dipolar kernel **K** in order to re-compute the form factor from the distance distribution. The kernel is constructed as in DeerAnalysis, where the maximal/minimal detectable distances are given by the time step and trace length. The kernel is then numerically computed using the same function as in DeerAnalysis.

```
%Construct kernel
rmin = (4*TimeStep*52.04/0.85)^(1/3);
rmax = 6*(length(TimeAxis)*TimeStep/2)^(1/3);
tmax = TimeStep*nPoints;
DistanceAxis = linspace(rmin,rmax,nPoints);
dr = mean(diff(DistanceAxis));
[Kernel,DistanceAxisKernel,~,U,sm,X,V,L] =  get_kernel(nPoints,TimeStep,rmin,
    rmax,2,struct());
Kernel = Kernel*dr;
```

Once the kernel has been computed, it can be used to obtain the noiseless dipolar evolution function. However, in order to directly obtain the form factor, we then add an offset. The kernel can then be applied to the distance distribution to obtain a form factor.

```
%Add offset to get form factors
for k=1:size(Kernel,1)
        Kernel(k,:) = Kernel(k,:) + Offset;
end
 %Construct extended time trace
TimeTrace = Kernel*ModelDistributions;
TimeTrace = TimeTrace/TimeTrace(1);
```

3.2. Background treatment

First we construct the background-free form factor. It only requires the addition of the noise and can be then regularized.

```
%CASE 1 (no background)
%-------------------------------------------------------------------------
%Add a.u. offset to form factor
%Prepare time trace
TimeTrace1 = LocalTimeTrace;
TimeTrace1 = TimeTrace1 + Noise;
%Regularize
[Distribution,~,FittedTimeTrace1,~,RegularizationParameter1] = regularizeDeer
    (TimeTrace1,TimeAxis,RegularizationMethod,options,[],KernelData);
%Normalize distribution to unity integral
Distribution1 = Distribution/sum(Distribution)/dr;
```

The background-divided case is obtained by, first multiplying the background function to the form factor and then adding the noise (the order is important to not damp the noise by the background multiplication). Once the noise is added, the background is divided and the resulting signal regularized.

```
%CASE 2 (background divided)
%------------------------------------------------------------------------
%Apply background
TimeTrace2 = LocalTimeTrace.*Background;
%Add noise
TimeTrace2 = TimeTrace2 + Noise;
%Correct background by division
TimeTrace2 = TimeTrace2./Background;
%Regularize
[Distribution,~,FittedTimeTrace2,~,RegularizationParameter2] = regularizeDeer
    (TimeTrace2,TimeAxis,RegularizationMethod,options,[],KernelData);
%Normalize distribution to unity integral
Distribution2 = Distribution/sum(Distribution)/dr;
```

Now we want to construct the signal for the $\mathbf{K_B}$ kernel method. As before, background and noise are added and left otherwise untreated. The dipolar kernel is modified according to the definition of the $\mathbf{K_B}$ kernel. This new kernel is then employed for the regularization with the untreated signal.

```
%CASE 3 (background in kernel)
%------------------------------------------------------------------------
%Apply background
TimeTrace3 = LocalTimeTrace.*Background;
%Add noise
TimeTrace3 = TimeTrace3 + Noise;
%Prepare kernel
KernelB = KernelData.Kernel;
KernelB = KernelB';
for k=1:size(KernelB,1)
KernelB(k,:)= Background'.*(KernelB(k,:));
end
KernelB = KernelB';
KernelBData = KernelData;
KernelBData.Kernel = KernelB;
%Regularize
[Distribution,~,FittedTimeTrace3,~,RegularizationParameter3] = regularizeDeer
    (TimeTrace3,TimeAxis,RegularizationMethod,options,Background,KernelBData);
%Normalize distribution to unity integral
Distribution3 = Distribution/sum(Distribution)/dr;
```

The background-subtracted signal is constructed as the divided one with the only difference being that the background is subtracted instead of divided.

```
%CASE 4 (background subtracted)
%------------------------------------------------------------------------
%Apply background
TimeTrace4 = LocalTimeTrace.*Background;
%Add noise
TimeTrace4 = TimeTrace4 + Noise;
%Prepare time trace
TimeTrace4 = TimeTrace4 + Offset - Background;
%Regularize
[Distribution,~,FittedTimeTrace4,~,RegularizationParameter4] = regularizeDeer
    (TimeTrace4,TimeAxis,RegularizationMethod,options,[],KernelData);
```

```
%Normalize distribution to unity integral
Distribution4 = Distribution/sum(Distribution)/dr;
```

For the $\mathbf{K}_{\sqrt{B}}$ kernel method, more careful processing is required. First the background and noise are added to the noiseless form factor. Then the square root of the background is divided from the signal. The basic dipolar kernel is then adapted to the definition of $\mathbf{K}_{\sqrt{B}}$. Now, before regularization, the optimal regularization parameter must be searched using the signal and kernel employed during the $\mathbf{K_B}$ method. Once it is found, the regularization is performed with the actual signal and the $\mathbf{K}_{\sqrt{B}}$ kernel.

```
%CASE 5 (background in kernel)
%------------------------------------------------------------------------
%Apply background
TimeTrace5 = LocalTimeTrace.*Background;
%Add noise
TimeTrace5 = TimeTrace5 + Noise;
%Correct square root of the background
TimeTrace5 = TimeTrace5./sqrt(Background);

%Get the regularization parameter using the signal and kernel of case 3 for
    the residual
options5 = options;
options5.KernelB = KernelB;
options5.TimeTraceB = TimeTrace3;

%Prepare kernel
KernelB2 = KernelData.Kernel;
KernelB2 = KernelB2';
for k=1:size(KernelB2,1)
KernelB2(k,:)= sqrt(Background)'.*(KernelB2(k,:));
end
KernelB2 = KernelB2';
KernelBData = KernelData;
KernelBData.Kernel = KernelB2;
%Search optimal regularization parameter with KB kernel and signal (from case
     3)
OptimalRegularizationParameter5 = selectionMethods('aic',TimeTrace5',KernelB2
    ,L,[],[],options5);
options5.ForcedRegularizationParameter = OptimalRegularizationParameter5;
options5.ForcedHuberParameter = 1;
%Regularize
[Distribution,~,FittedTimeTrace5,~,RegularizationParameter5] = regularizeDeer
    (TimeTrace5,TimeAxis,RegularizationMethod,options5,Background,KernelBData)
    ;
%Normalize distribution to unity integral
Distribution5 = Distribution/sum(Distribution)/dr;
```